

Trabajo Práctico 1:

Conjunto de instrucciones MIPS

Fabrizio Cozza, *Padrón Nro. 97.402*
fabrizio.cozza@gmail.com

Kevin Cajachuán, *Padrón Nro. 98.725*
kevincajachuan@hotmail.com

Luciano Giannotti, *Padrón Nro. 97.215*
luciano_giannotti@hotmail.com.ar

1er. Cuatrimestre de 2018
66.20 Organización de Computadoras – Práctica Martes
Facultad de Ingeniería, Universidad de Buenos Aires

1. Objetivos

Este Trabajo Práctico tiene el fin de ayudarnos a familiarizarnos con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descrito en la siguiente sección.

2. Programa

El software de este trabajo está escrito en su mayoría en lenguaje C y permite dibujar **Julia Sets** o **Conjuntos de Julia** según los parámetros que le pasamos por línea de comando. Estos parámetros son la región del plano complejo: delimitada por un centro, un ancho y un alto; una semilla que afectará el cálculo para cada píxel; la resolución y la salida ya sea por pantalla o por archivo.

La función en la cuál se encuentra la lógica de cómputo del fractal está escrita en MIPS con el fin de tener soporte nativo para NetBSD.

El formato a usar es PGM o *portable gray format*, que resulta útil para describir imágenes digitales en escala de grises.

3. Implementación

Una vez recibidos los parámetros, para dibujar el Julia Set el programa convierte cada píxel de la ventana a un punto en el plano complejo. A ese punto se lo eleva al cuadrado y le suma la semilla mencionada en la sección anterior. Esto se repite hasta que el valor absoluto del resultado sea menor a 2, en cuyo caso se toma la cantidad de iteraciones y se imprime en el archivo PGM, representando el nivel de blanco de ese píxel.

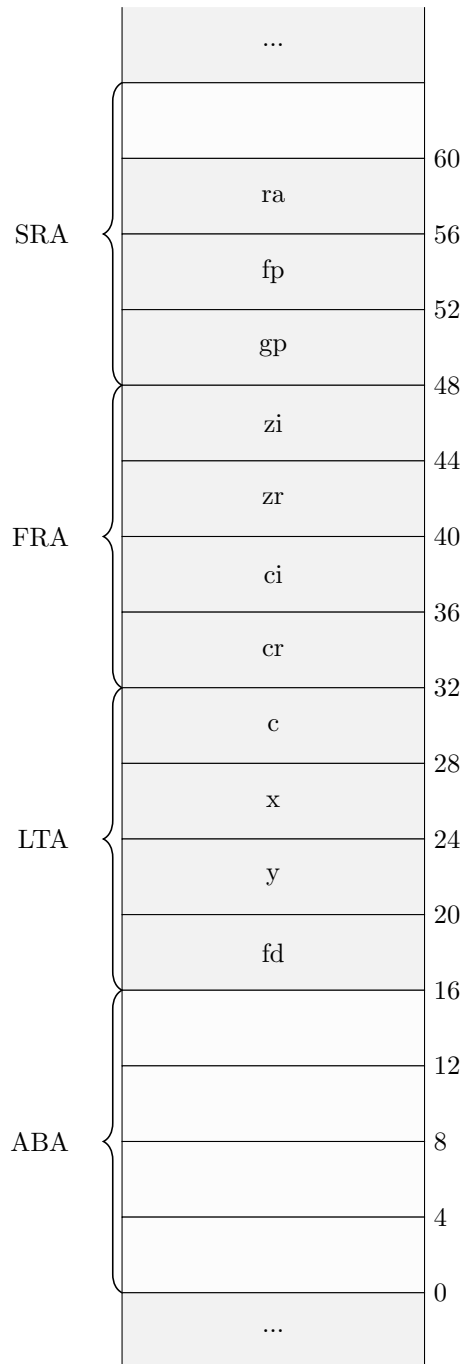
3.1. Funciones implementadas

3.1.1. mips32_plot

Esta función es la que se encarga de hacer los cálculos, para luego poder ir imprimiendo el valor de cada píxel en un archivo.

El único parámetro que recibe es un struct definido como *param_t* en el que se encuentran todos los datos necesarios para que la función realice su tarea, los cuales se obtienen de los parámetros pasados por el usuario por línea de comandos.

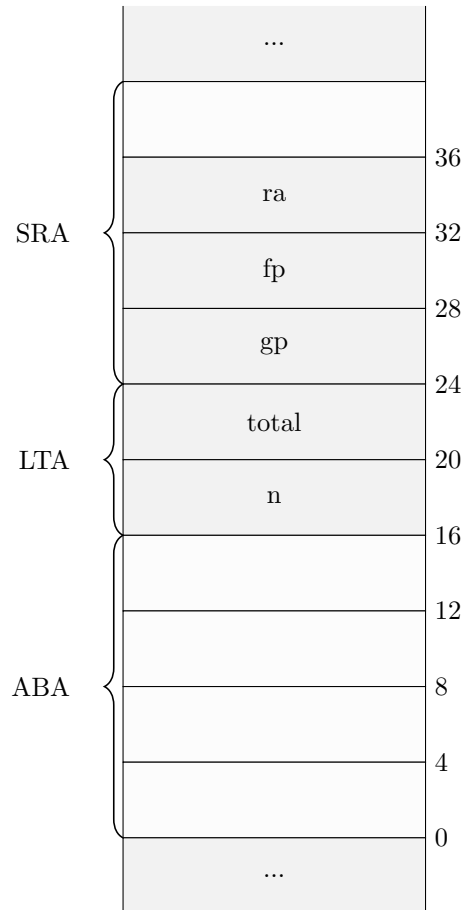
Esta función no devuelve nada ya que solo se dedica a hacer cálculos e imprimir. El stack frame de esta función se muestra a continuación, en el cuál en cada dirección de memoria se indica qué variable está guardada.



3.1.2. my_fprintf

Esta función imprime en un archivo llamando a la syscall *write*, por lo que los parámetros que recibe son los mismo que recibe esta syscall: file descriptor del archivo, lo que se quiere escribir, y cuánto se quiere escribir. Finalmente devuelve la cantidad de bytes que se escribió la igual que la syscall.

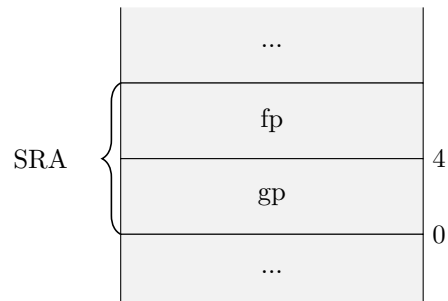
El stack frame de esta función se muestra a continuación.



3.1.3. my_strlen

Esta función calcula la longitud de una cadena, recibiendo como parámetro la cadena y devolviendo la longitud.

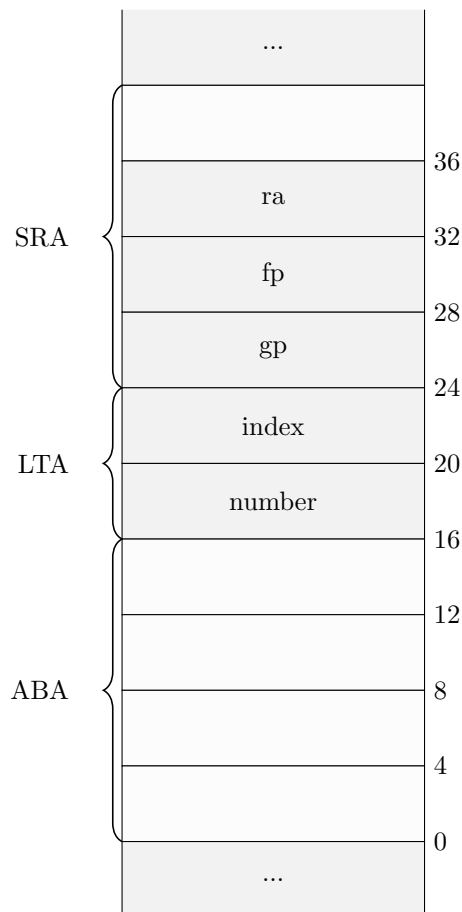
El stack frame de la función se muestra a continuación. Al ser una función *leaf*, no fue necesario tener un ABA ni salvar el registro *ra*.



3.1.4. `int_to_str`

Esta función convierte un número a una cadena, llamando para realizar esto a dos funciones: *dig_to_char* y *put_end*. Lo único que recibe la función es el número que se quiere convertir y no devuelve nada.

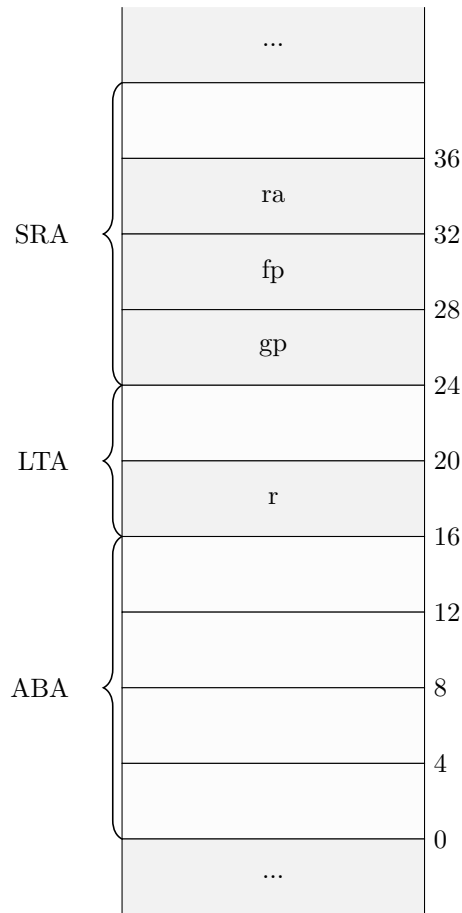
El stack frame se muestra a continuación.



3.1.5. `dig_to_char`

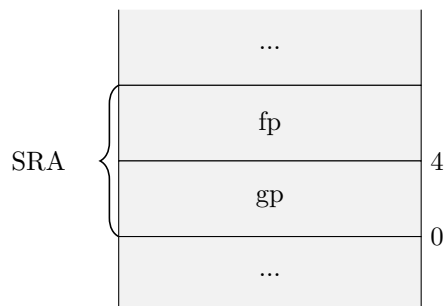
Esta función recibe un número que se quiere convertir a cadena, un array para ir guardando los caracteres de cada dígito del número y el índice actual en el que hay que ir guardando cada caracter. Esta función se llama recursivamente para ir obteniendo los dígitos del número e ir guardandolos en el array. Como esto es lo único que realiza, la función no devuelve nada.

A continuación se muestra el stack frame de esta función.



3.1.6. `put_end`

Esta función lo único que hace es poner el caracter `\0` al final del array en que se guardaron los dígitos del número, para poder imprimirlo más adelante. Por esta razón esta función recibe como parámetros el array y el índice en el que se tiene que guardar el caracter `\0` y no devuelve ningún valor. El stack frame se muestra a continuación. Por la misma razón que en la función *my_strlen*, no es necesario tener un ABA ni salvar *ra*.



4. Pruebas

Como estamos probando imagenes, las pruebas las realizamos a ojo, comparandolas con las imagenes provistas como ejemplo en el enunciado y otras obtenidas con un generador online (<http://usefuljs.net/fractals/>). Las imagenes de prueba se generaron corriendo el archivo **test.sh**, pero en las siguientes subsecciones se mostrará el código que genera cada imagen. Además se compararán los resultados obtenidos por el método genérico (usando la función *c_plot*) y el método MIPS32 (usando la función *mips32_plot* escrita en assembly para este TP).

5. Compilación

Como hay un archivo *Makefile* que realiza la compilación, el único comando que hay que escribir en la terminal para compilarlo es:

```
$ make
```

6. Código S

En esta sección se muestra el código de la función *mips32_plot*.

```
#include <sys/syscall.h>
#include <mips/regdef.h>

        .text
        .abicalls
        .align 2
        .globl mips32_plot
        .ent mips32_plot
mips32_plot:
        .frame $fp, 64, ra
        .set noreorder
        .cpload t9
        .set reorder
        subu sp, sp, 64
        .cpstore 48
        sw $fp, 52(sp)
        sw ra, 56(sp)
        move $fp, sp
        sw a0, 64($fp)
        lw t0, 44(a0)
        lh t0, 14(t0)
        sw t0, 16($fp)
        la a0, head_pgm
        jal my_strlen
        move a2, v0
        lw a0, 16($fp)
        la a1, head_pgm
        jal my_fprintf
        lw t0, 64($fp)
        lw a0, 32(t0)
        jal int_to_str
        la a0, number
        jal my_strlen
        move a2, v0
        lw a0, 16($fp)
        la a1, number
        jal my_fprintf
        lw a0, 16($fp)
        la a1, new_line
        li a2, 1
        jal my_fprintf

        # Fin de creacion de SF
        # Obtengo dir de fp en t0
        # Obtengo fd
        # Salvo fd

        # Cargo len(head_pgm)
        # Cargo fd
        # Cargo head_pgm
        # Imprimo "p2"

        # Cargo x_res
        # x_res -> str

        # Cargo len(number)
        # Cargo fd
        # Cargo number
        # Imprimo "x_res"
        # Cargo fd
        # Cargo "\n"
        # Cargo len("\n")
        # Imprimo "\n"
```

```

lw t0, 64($fp)
lw a0, 36(t0)
jal int_to_str
la a0, number
jal my_strlen
move a2, v0
lw a0, 16($fp)
la a1, number
jal my_fprintf
lw a0, 16($fp)
la a1, new_line
li a2, 1
jal my_fprintf
lw t0, 64($fp)
lw a0, 40(t0)
subu a0, a0, 1
jal int_to_str
la a0, number
jal my_strlen
move a2, v0
lw a0, 16($fp)
la a1, number
jal my_fprintf
lw a0, 16($fp)
la a1, new_line
li a2, 1
jal my_fprintf

move t0, zero
sw t0, 20($fp)
lw t1, 64($fp)
lwc1 $f20, 4(t1)
swc1 $f20, 36($fp)
b test_for1

for1:
move t0, zero
sw t0, 24($fp)
lw t1, 64($fp)
lwc1 $f20, 0(t1)
swc1 $f20, 32($fp)
b test_for2

for2:
lwc1 $f20, 32($fp)
swc1 $f20, 40($fp)
lwc1 $f20, 36($fp)
swc1 $f20, 44($fp)

move t0, zero
sw t0, 28($fp)
b test_for3

for3:
lwc1 $f20, 40($fp)
lwc1 $f22, 44($fp)
mul.s $f24, $f20, $f20
mul.s $f26, $f22, $f22
add.s $f28, $f24, $f26
li.s $f30, 4
c.le.s $f30, $f28
bclt print_pixel

lw t1, 64($fp)
lwc1 $f30, 24(t1)
add.s $f30, $f30, $f24
sub.s $f30, $f30, $f26
lw t1, 64($fp)
lwc1 $f24, 28(t1)
mul.s $f26, $f20, $f22
li.s $f20, 2
mul.s $f26, $f26, $f20
add.s $f22, $f24, $f26
swc1 $f30, 40($fp)
swc1 $f22, 44($fp)

lw t0, 28($fp)

# Cargo y_res
# y_res -> str

# Cargo len(number)
# Cargo fd
# Cargo number
# Imprimo "y_res"
# Cargo fd
# Cargo "\n"
# Cargo len("\n")
# Imprimo "\n"

# Cargo shades
# shades-1
# shades-1 -> str

# Cargo len(shades-1)
# Cargo fd
# Cargo number
# Imprimo "shades-1"
# Cargo fd
# Cargo "\n"
# Cargo len("\n")
# Imprimo "\n"

# Salvo y

# Obtengo UL_im
# Salvo ci = UL_im

# Salvo x

# Obtengo UL_re
# Salvo cr = UL_im

# Obtengo cr
# Salvo zr = cr
# Obtengo ci
# Salvo zi = ci

# Salvo c

# Obtengo zr
# Obtengo zi
# zr * zr
# zi * zi
# zr * zr + zi * zi

# 4 <= absz
# break

# Obtengo s_re
# s_re + zr * zr
# s_re + zr * zr - zi * zi

# Obtengo s_im
# zr * zi

# zr * zi * 2
# s_im + zr * zi * 2
# zr = tr
# zi = ti

# Obtengo c

```



```

        addiu t0, t0, 1
        sw t0, 28($fp)
test_for3:
        lw t0, 64($fp)
        lw t0, 40(t0)
        lw t1, 28($fp)
        subu t2, t0, t1
        bgtz t2, for3

print_pixel:
        lw a0, 28($fp)
        jal int_to_str
        la a0, number
        jal my_strlen
        move a2, v0
        lw a0, 16($fp)
        la a1, number
        jal my_fprintf
        lw a0, 16($fp)
        la a1, new_line
        li a2, 1
        jal my_fprintf

        lw t0, 24($fp)
        addiu t0, t0, 1
        sw t0, 24($fp)
        lwc1 $f20, 32($fp)
        lw t2, 64($fp)
        lwc1 $f22, 16(t2)
        add.s $f20, $f20, $f22
        swc1 $f20, 32($fp)
test_for2:
        lw t0, 64($fp)
        lw t0, 32(t0)
        lw t1, 24($fp)
        subu t2, t0, t1
        bgtz t2, for2

        lw t0, 20($fp)
        addiu t0, t0, 1
        sw t0, 20($fp)
        lwc1 $f20, 36($fp)
        lw t2, 64($fp)
        lwc1 $f22, 20(t2)
        sub.s $f20, $f20, $f22
        swc1 $f20, 36($fp)
test_for1:
        lw t0, 64($fp)
        lw t0, 36(t0)
        lw t1, 20($fp)
        subu t2, t0, t1
        bgtz t2, for1

return:
        lw ra, 56(sp)
        lw $fp, 52(sp)
        lw gp, 48(sp)
        addu sp, sp, 64
        jr ra
        .end mips32_plot
        .size mips32_plot, .-mips32_plot

        .ent int_to_str
int_to_str:
        .frame $fp, 40, ra
        .set noreorder
        .cpload t9
        .set reorder
        subu sp, sp, 40
        .cprestore 24
        sw $fp, 28(sp)
        sw ra, 32(sp)
        move $fp, sp
        sw a0, 40($fp)

```

++c
Salvo c

Obtengo shades
Obtengo c
shades - c

Obtengo c
c -> str

Cargo len(number)
Cargo fd
Cargo number
Imprimo "c"
Cargo fd
Cargo "\n"
Cargo len("\n")
Imprimo "\n"

Obtengo x
++x
Salvo x
Obtengo cr

Obtengo d_re
cr += d_re
Salvo cr

Obtengo x_res
Obtengo x
x_res - x

Obtengo y
++y
Salvo y
Obtengo ci

Obtengo d_im
ci -= d_im
Salvo ci

Obtengo y_res
Obtengo y
y_res - y

Fin de creacion de SF

```

        la a1, number
        la a2, index
        jal dig_to_char
        la a0, number
        la a1, index
        jal put_end
        lw ra, 32(sp)
        lw $fp, 28(sp)
        lw gp, 24(sp)
        addu sp, sp, 40
        jr ra
        .end int_to_str
        .size int_to_str, .-int_to_str

        .ent dig_to_char
dig_to_char:
        .frame $fp, 40, ra
        .set noreorder
        .cplod t9
        .set reorder
        subu sp, sp, 40
        .cpstore 24
        sw $fp, 28(sp)
        sw ra, 32(sp)
        move $fp, sp
        sw a0, 40($fp)
        sw a1, 44($fp)
        sw a2, 48($fp)
        beqz a0, return_dig
        remu t0, a0, 10
        sw t0, 16($fp)
        divu a0, a0, 10
        jal dig_to_char
        lw t0, 16($fp)
        lb t1, 0(a2)
        addu t2, a1, t1
        addiu t0, t0, 48
        sb t0, 0(t2)
        addiu t1, t1, 1
        sb t1, 0(a2)
return_dig:
        lw ra, 32(sp)
        lw $fp, 28(sp)
        lw gp, 24(sp)
        addu sp, sp, 40
        jr ra
        .end dig_to_char
        .size dig_to_char, .-dig_to_char

        .ent put_end
put_end:
        .frame $fp, 8, ra
        .set noreorder
        .cplod t9
        .set reorder
        subu sp, sp, 8
        .cpstore 0
        sw $fp, 4(sp)
        move $fp, sp
        sw a0, 8($fp)
        sw a1, 12($fp)
        lb t0, 0(a1)
        addu t0, a0, t0
        sb zero, 0(t0)
        sb zero, 0(a1)
        lw $fp, 4(sp)
        lw gp, 0(sp)
        addu sp, sp, 8
        jr ra
        .end put_end
        .size put_end, .-put_end

        .ent my_strlen
my_strlen:
        # Cargo array
        # Cargo indice
        # Convierto digitos a char
        # Cargo array
        # Cargo indice
        # \0 al final

        # Fin de creacion de SF

        # r = n % 10

        # ascii del numero

        # Fin de creacion de SF

        # Reinicio indice

```

```

        .frame $fp, 8, ra
        .set noreorder
        .cplod t9
        .set reorder
        subu sp, sp, 8
        .cpstore 0
        sw $fp, 4(sp)
        move $fp, sp
        sw a0, 8($fp)
        move t0, zero
        b test_end
increment:
        addiu t0, t0, 1
test_end:
        addu t1, t0, a0
        lb t1, 0(t1)
        bnez t1, increment
        increment
        move v0, t0
        lw $fp, 4(sp)
        lw gp, 0(sp)
        addu sp, sp, 8
        jr ra
        .end my_strlen
        .size my_strlen, .-my_strlen

        .ent my_fprintf
my_fprintf:
        .frame $fp, 40, ra
        .set noreorder
        .cplod t9
        .set reorder
        subu sp, sp, 40
        .cpstore 24
        sw $fp, 28(sp)
        sw ra, 32(sp)
        move $fp, sp
        sw a0, 40($fp)
        sw a1, 44($fp)
        sw a2, 48($fp)
        move t0, zero
        move t1, zero
        sw t0, 16($fp)
        sw t1, 20($fp)
        b test_write
add_total:
        lw t1, 20($fp)
        lw t0, 16($fp)
        addu t1, t1, t0
        sw t0, 16($fp)
        sw t1, 20($fp)
test_write:
        lw a0, 40($fp)
        lw t3, 44($fp)
        lw t2, 48($fp)
        lw t1, 20($fp)
        lw t0, 16($fp)
        addu a1, t3, t1
        subu a2, t2, t1
        li v0, SYS_write
        syscall
        move t0, v0
        sw t0, 16($fp)
        bgtz t0, add_total
        lw v0, 20($fp)
        lw ra, 32(sp)
        lw $fp, 28(sp)
        lw gp, 24(sp)
        addu sp, sp, 40
        jr ra
        .end my_fprintf
        .size my_fprintf, .-my_fprintf

        .rdata

```

Fin de creacion de SF
i = 0

i++

if(s[i] != 0) ->

Fin de creacion de SF
n = 0
total = 0
n -> SF
total -> SF

Cargo fd

Cargo dir + total
Cargo len - total

Devuelvo el total

```
        .align 2
msgs:    .word head_pgm, new_line
        .align 0
head_pgm: .asciiiz "P2\n"
new_line: .asciiiz "\n"

        .data
number:  .space 11
index:   .byte 0
```

7. Bibliografía

1. GXemul.
<http://gavare.se/gxemul/>.
2. The NetBSD project.
<http://www.netbsd.org/>.
3. Conjunto de Julia
http://es.wikipedia.org/wiki/Conjunto_de_Julia (Wikipedia).
4. PGM format specification.
<http://netpbm.sourceforge.net/doc/pgm.html>.
5. Generador de fractales.
<http://usefuljs.net/fractals/>
6. GIMP.
<https://www.gimp.org/>
7. System V Application Binary Interface.
<http://math-atlas.sourceforge.net/devel/assembly/mipsabi32.pdf>